

Distributed File Systems

B.Ramamurthy

Introduction

- ◆ Distributed file systems support the sharing of information in the form of files throughout the intranet.
- ◆ A distributed file system enables programs to store and access remote files exactly as they do on local ones, allowing users to access files from any computer on the intranet.
- ◆ Recent advances in higher bandwidth connectivity of switched local networks and disk organization have lead high performance and highly scalable file systems.

Storage systems and their properties

	Sharing	Persis- tence	Distributed cache/replicas	Consistency maintenance	Example
Main memory	×	×	×		RAM
File system	×	✓	×		UNIX file system
Distributed file system	✓	✓	✓	✓	Sun NFS
Web	✓	✓	✓	×	Web server
Distributed shared memory	✓	×	✓	✓	Ivy (Ch. 16)
Remote objects (RMI/ORB)	✓	×	×		CORBA
Persistent object store	✓	✓	×		CORBA Persistent Object Service
Persistent distributed object store	✓	✓	✓	✓	PerDis, Khazana

File system modules

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

File attribute record structure

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list (ACL)

UNIX file system operations

<code>filedes = open(name, mode)</code>	Opens an existing file with the given <i>name</i> .
<code>filedes = creat(name, mode)</code>	Creates a new file with the given <i>name</i> . Both operations deliver a file descriptor referencing the open file. The <i>mode</i> is <i>read</i> , <i>write</i> or both.
<code>status = close(filedes)</code>	Closes the open file <i>filedes</i> .
<code>count = read(filedes, buffer, n)</code>	Transfers <i>n</i> bytes from the file referenced by <i>filedes</i> to <i>buffer</i> .
<code>count = write(filedes, buffer, n)</code>	Transfers <i>n</i> bytes to the file referenced by <i>filedes</i> from <i>buffer</i> . Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
<code>pos = lseek(filedes, offset, whence)</code>	Moves the read-write pointer to offset (relative or absolute, depending on <i>whence</i>).
<code>status = unlink(name)</code>	Removes the file <i>name</i> from the directory structure. If the file has no other names, it is deleted.
<code>status = link(name1, name2)</code>	Adds a new name (<i>name2</i>) for a file (<i>name1</i>).
<code>status = stat(name, buffer)</code>	Gets the file attributes for file <i>name</i> into <i>buffer</i> .

Distributed File System Requirements

- ◆ Many of the requirements of distributed services were lessons learned from distributed file service.
- ◆ First needs were: access transparency and location transparency.
- ◆ Later on, performance, scalability, concurrency control, fault tolerance and security requirements emerged and were met in the later phases of DFS development.

9/28/2004

B.Ramamurthy

7

Transparency

- ◆ Access transparency: Client programs should be unaware of the the distribution of files.
- ◆ Location transparency: Client program should see a uniform namespace. Files should be able to be relocated without changing their path name.
- ◆ Mobility transparency: Neither client programs nor system admin program tables in the client nodes should be changed when files are moved either automatically or by the system admin.
- ◆ Performance transparency: Client programs should continue to perform well on load within a specified range.
- ◆ Scaling transparency: increase in size of storage and network size should be transparent.

9/28/2004

B.Ramamurthy

8

Other Requirements

- ◆ Concurrent file updates is protected (record locking).
- ◆ File replication to allow performance.
- ◆ Hardware and operating system heterogeneity.
- ◆ Fault tolerance
- ◆ Consistency : Unix uses on-copy update semantics. This may be difficult to achieve in DFS.
- ◆ Security
- ◆ Efficiency

9/28/2004

B.Ramamurthy

9

General File Service Architecture

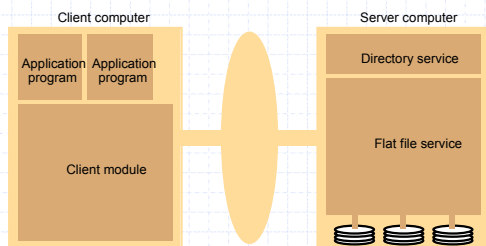
- ◆ The responsibilities of a DFS are typically distributed among three **modules**:
 - **Client module** which emulates the conventional file system interface
 - **Server modules(2)** which perform operations for clients on **directories** and on **files**.
- ◆ Most importantly this architecture enables stateless implementation of the server modules.

9/28/2004

B.Ramamurthy

10

File service architecture



9/28/2004

B.Ramamurthy

11

Flat file service Interface

Read(FileId, i, n) -> Data If $1 \leq i \leq \text{Length}(\text{File})$: Reads a sequence of up to n items from a file starting at item i and returns it in *Data*.
 — throws *BadPosition*

Write(FileId, i, Data) If $1 \leq i \leq \text{Length}(\text{File})+1$: Writes a sequence of *Data* to a file, starting at item i , extending the file if necessary.
 — throws *BadPosition*

Create() -> FileId Creates a new file of length 0 and delivers a UFID for it.

Delete(FileId) Removes the file from the file store.

GetAttributes(FileId) -> Attr Returns the file attributes for the file.

SetAttributes(FileId, Attr) Sets the file attributes (only those attributes that are not shaded in).

Primary operations are reading and writing.

9/28/2004

B.Ramamurthy

12

Directory service Interface

<code>Lookup(Dir, Name) -> FileId</code> — throws <code>NotFound</code>	Locates the text name in the directory and returns the relevant UFID. If <code>Name</code> is not in the directory, throws an exception.
<code>AddName(Dir, Name, File)</code> — throws <code>NameDuplicate</code>	If <code>Name</code> is not in the directory, adds <code>(Name, File)</code> to the directory and updates the file's attribute record. If <code>Name</code> is already in the directory: throws an exception.
<code>UnName(Dir, Name)</code> — throws <code>NotFound</code>	If <code>Name</code> is in the directory: the entry containing <code>Name</code> is removed from the directory. If <code>Name</code> is not in the directory: throws an exception.
<code>GetNames(Dir, Pattern) -> NameSeq</code>	Returns all the text names in the directory that match the regular expression <code>Pattern</code> .

Primary purpose is to provide a service for translation text names to UFIDs.

9/28/2004

B.Ramamurthy

13

Case Studies in DFS

- ◆ We will look into architecture and operation of SUN's Network File System (NFS) and CMU's Andrew File System (AFS).

9/28/2004

B.Ramamurthy

14

Network File System

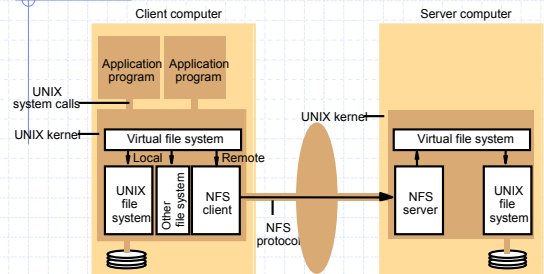
- ◆ The Network File System (NFS) was developed to allow machines to mount a disk partition on a remote machine as if it were on a local hard drive. This allows for fast, seamless sharing of files across a network.

9/28/2004

B.Ramamurthy

15

NFS architecture



9/28/2004

B.Ramamurthy

16

NFS server operations (simplified) – 1

<code>lookup(dirfh, name) -> fh, attr</code>	Returns file handle and attributes for the file <code>name</code> in the directory <code>dirfh</code> .
<code>create(dirfh, name, attr) -> newfh, attr</code>	Creates a new file name in directory <code>dirfh</code> with attributes <code>attr</code> and returns the new file handle and attributes.
<code>remove(dirfh, name) status</code>	Removes file name from directory <code>dirfh</code> .
<code>getattr(fh) -> attr</code>	Returns file attributes of file <code>fh</code> . (Similar to the UNIX <code>stat</code> system call.)
<code>setattr(fh, attr) -> attr</code>	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.
<code>read(fh, offset, count) -> attr, data</code>	Returns up to <code>count</code> bytes of data from a file starting at <code>offset</code> . Also returns the latest attributes of the file.
<code>write(fh, offset, count, data) -> attr</code>	Writes <code>count</code> bytes of data to a file starting at <code>offset</code> . Returns the attributes of the file after the write has taken place.
<code>rename(dirfh, name, todirfh, toname) -> status</code>	Changes the name of file name in directory <code>dirfh</code> to <code>toname</code> in directory <code>toaddirfh</code> .
<code>link(newdirfh, newname, dirfh, name) -> status</code>	Creates an entry <code>newname</code> in the directory <code>newdirfh</code> which refers to file name in the directory <code>dirfh</code> .

Continues on next slide

9/28/2004

B.Ramamurthy

17

NFS server operations (simplified) – 2

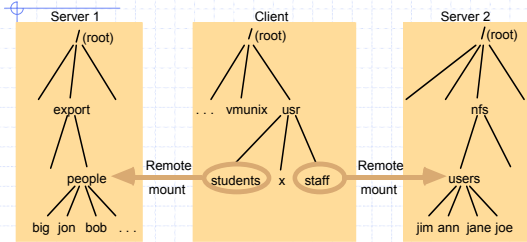
<code>symlink(newdirfh, newname, string) -> status</code>	Creates an entry <code>newname</code> in the directory <code>newdirfh</code> of type symbolic link with the value <code>string</code> . The server does not interpret the <code>string</code> but makes a symbolic link file to hold it.
<code>readlink(fh) -> string</code>	Returns the string that is associated with the symbolic link file identified by <code>fh</code> .
<code>mkdir(dirfh, name, attr) -> newfh, attr</code>	Creates a new directory <code>name</code> with attributes <code>attr</code> and returns the new file handle and attributes.
<code>rmdir(dirfh, name) -> status</code>	Removes the empty directory <code>name</code> from the parent directory <code>dirfh</code> . Fails if the directory is not empty.
<code>readdir(dirfh, cookie, count) -> entries</code>	Returns up to <code>count</code> bytes of directory entries from the directory <code>dirfh</code> . Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a <code>cookie</code> . The <code>cookie</code> is used in subsequent <code>readdir</code> calls to start reading from the following entry. If the value of <code>cookie</code> is 0, reads from the first entry in the directory.
<code>stats(fh) -> fsstats</code>	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file <code>fh</code> .

9/28/2004

B.Ramamurthy

18

Local and remote file systems accessible on an NFS client



Note: The file system mounted at `/usr/students` in the client is actually the sub-tree located at `/export/people` in Server 1. The file system mounted at `/usr/staff` in the client is actually the sub-tree located at `/nfs/users` in Server 2.

9/28/2004

B.Ramamurthy

19

NFS Revisited

- ◆ From A.Tannenbaum's text
- ◆ Three aspects of NFS are of interest: the architecture, the protocol, and the implementation.

9/28/2004

B.Ramamurthy

20

NFS Architecture

- ◆ Allows an arbitrary collection of clients and servers to share a common file system.
- ◆ In many cases all servers and clients are on the same LAN but this is not required.
- ◆ NFS allows every machine to be a client and server at the same time.
- ◆ Each NFS server exports one or more directories for access by remote clients.

9/28/2004

B.Ramamurthy

21

NFS Protocol

- ◆ One of the goals of NFS is to support a heterogeneous system, with clients and servers running different operating systems on different hardware. It is essential the interface between clients and server be well defined.
- ◆ NFS accomplishes this goal by defining two client-server protocols: one for handling mounting and another for directory and file access.
- ◆ Protocol defines requests by clients and responses by servers.

9/28/2004

B.Ramamurthy

22

Mounting

- ◆ Client requests a directory structure to be mounted, if the path is legal the server returns file handle to the client.
- ◆ Or the mounting can be automatic by placing the directories to be mounted in the `/etc/rc`: automounting.

9/28/2004

B.Ramamurthy

23

File Access

- ◆ NFS supports most unix operations except open and close. This is to satisfy the "statelessness" on the server end. Server need not keep a list of open connections. See the operations listed in slides 17, 18.
- ◆ (On the other hand consider your database connection... you create an object, connection is opened etc.)

9/28/2004

B.Ramamurthy

24

Implementation

- ◆ After the usual system call layer, NFS specific layer Virtual File System (VFS) maintains an entry per file called vnode (virtual I-node) for every open file.
- ◆ Vnode indicate whether a file is local or remote.
 - For remote files extra info is provided.
 - For local file, file system and I-node are specified.
 - Lets see how to use v-nodes using a mount, open, read system calls from a client application.

9/28/2004

B.Ramamurthy

25

Vnode use

- ◆ To mount a remote file system, the sys admin (or /etc/rc) calls the mount program specifying the remote directory, local directory in which to be mounted, and other info.
- ◆ If the remote directory exist and is available for mounting, mount system call is made.
- ◆ Kernel constructs vnode for the remote directory and asks the NFS-client code to create a r-node (remote I-node) in its internal tables. V-node in the client VFS will point to local I-node or this r-node.

9/28/2004

B.Ramamurthy

26

Remote File Access

- ◆ When a remote file is opened by the client, it locates the r-node.
- ◆ It then asks NFS Client to open the file. NFS file looks up the path in the remote file system and return the file handle to VFS tables.
- ◆ The caller (application) is given a file descriptor for the remote file. No table entries are made on the server side.
- ◆ Subsequent reads will invoke the remote file, and for efficiency sake the transfers are usually in large chunks (8K).

9/28/2004

B.Ramamurthy

27

Server Side of File Access

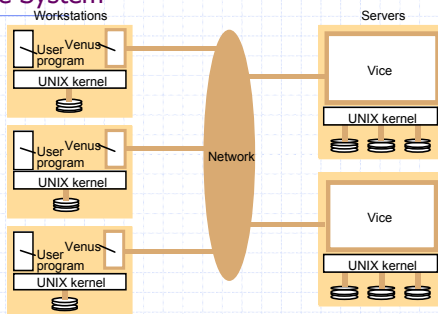
- ◆ When the request message arrives at the NFS server, it is passed to the VFS layer where the file is probably identified to be a local or remote file.
- ◆ Usually a 8K chunk is returned. **Read ahead and caching** are used to improve efficiency.
- ◆ Cache: server side for disk accesses, client side for I-nodes and another for file data.
- ◆ Of course this leads to cache consistency and security problem which ties us into other topics we are discussing.

9/28/2004

B.Ramamurthy

28

Distribution of processes in the Andrew File System



9/28/2004

B.Ramamurthy

29

Summary

- ◆ Study Andrew Files System (AFS): how?
- ◆ Architecture
- ◆ APIs for operations
- ◆ Protocols for operations
- ◆ Implementation details

9/28/2004

B.Ramamurthy

30